**M. Ali. Arslan**

**19F-0348**

**Assignment-01**

**Link List**

**Task # 01**

#include<iostream>

#include<iomanip>

using namespace std;

struct node

{

int data;

node\* previous\_node;

node\* next\_node;

};

class double\_list {

public:

node\* head;

node\* tail;

int count;

double\_list()

{

head = tail = NULL;

count = 1;

}

void insertion(int value)

{

node\* new\_node = new node;

new\_node->data = value;

new\_node->next\_node = NULL;

new\_node->previous\_node = NULL;

if (head == NULL)

{

head = new\_node;

tail = new\_node;

}

else

{

tail->next\_node = new\_node;

new\_node->previous\_node = tail;

tail = tail->next\_node;

}

count++;

node\* temp\_1 = head;

node\* temp\_2 = head->next\_node;

while (temp\_1->next\_node != NULL)

{

while (temp\_2 != NULL)

{

if (temp\_1->data > temp\_2->data)

{

int temp\_3 = temp\_2->data;

temp\_2->data = temp\_1->data;

temp\_1->data = temp\_3;

}

temp\_2 = temp\_2->next\_node;

}

temp\_1 = temp\_1->next\_node;

temp\_2 = temp\_1->next\_node;

}

}

void display()

{

node\* temp\_1 = head;

while (temp\_1 != NULL)

{

cout << " " << temp\_1->data;

temp\_1 = temp\_1->next\_node;

}

}

void deletion(int node\_number)

{

if (node\_number == 1)

{

node\* temp\_1 = head;

head->next\_node = temp\_1->next\_node;

head = head->next\_node;

head->previous\_node = NULL;

delete temp\_1;

}

else if (node\_number > 1 && node\_number < count - 1)

{

node\* temp\_2 = head;

for (int i = 1; i < node\_number; i++)

{

temp\_2 = temp\_2->next\_node;

}

temp\_2->previous\_node->next\_node = temp\_2->next\_node;

temp\_2->next\_node->previous\_node = temp\_2->previous\_node;

delete temp\_2;

}

else

{

node\* temp\_3 = tail;

tail->previous\_node->next\_node = NULL;

tail = tail->previous\_node;

delete temp\_3;

}

}

};

int main()

{

double\_list a;

bool check = true;

int choice, value\_1;

while (check)

{

cout << "1) Insert value" << endl;

cout << "2) Delete value" << endl;

cout << "3) Display the values" << endl;

cout << "0) Exit" << endl;

cout << "Enter choice: ";

cin >> choice;

cout << endl;

switch (choice)

{

case 1:

cout << "Enter a vlaue: ";

cin >> value\_1;

a.insertion(value\_1);

cout << endl;

break;

case 2:

value\_1 = 0;

cout << "Enter node which you want to delete: ";

cin >> value\_1;

a.deletion(value\_1);

cout << "Value has been deleted from linklist " << endl;

cout << endl;

break;

case 3:

cout << "Link List is : ";

a.display();

cout << endl;

break;

case 0:

exit(0);

break;

default:

cout <<"Invalid Choice " << endl;

break;

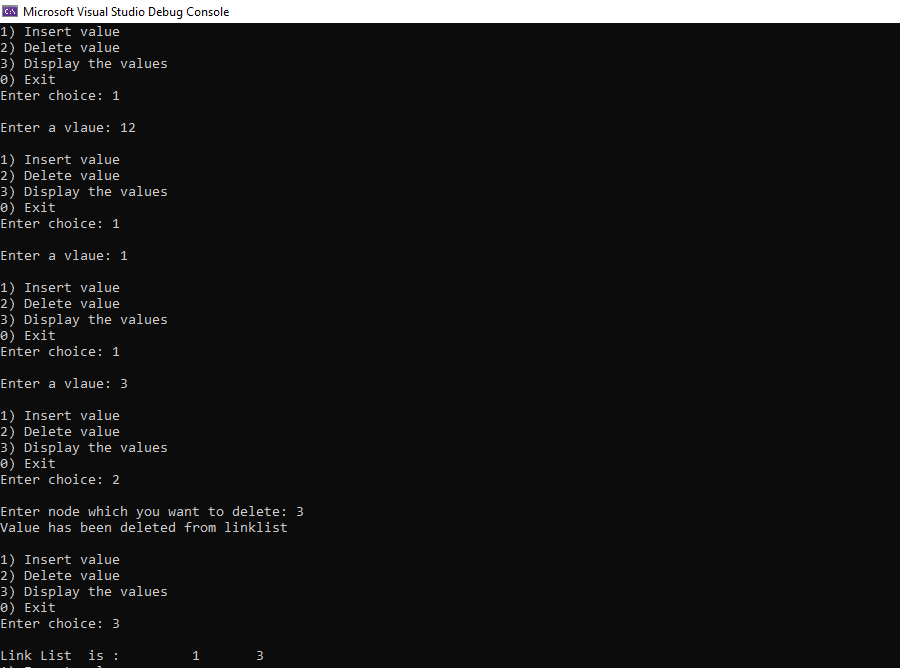
}

}

system("pause");

return 0;

}



**Task # 02**

#include<iostream>

#include<iomanip>

using namespace std;

struct node

{

node\* next;

int data;

};

class LinkedList

{

public:

node\* head;

node\* tail;

LinkedList()

{

head = tail = NULL;

}

void insertion(int value)

{

node\* new\_node = new node;

new\_node->data = value;

new\_node->next = NULL;

if (head == NULL)

{

head = tail = new\_node;

}

else

{

tail->next = new\_node;

tail = tail->next;

}

}

void display()

{

node\* temp\_1 = NULL;

temp\_1 = head;

while (temp\_1 != NULL)

{

cout << " " << temp\_1->data << "->";

temp\_1 = temp\_1->next;

}

}

void swap(int first\_node\_value, int second\_node\_value)

{

node\* first\_ptr = head;

node\* previous\_of\_first\_ptr = NULL;

while (first\_ptr != NULL && first\_ptr->data != first\_node\_value)

{

previous\_of\_first\_ptr = first\_ptr;

first\_ptr = first\_ptr->next;

}

node\* second\_ptr = head;

node\* previous\_of\_second\_ptr = NULL;

while (second\_ptr != NULL && second\_ptr->data != second\_node\_value)

{

previous\_of\_second\_ptr = second\_ptr;

second\_ptr = second\_ptr->next;

}

if (previous\_of\_first\_ptr == NULL)

{

head = second\_ptr;

}

else

{

previous\_of\_first\_ptr->next = second\_ptr;

}

if (previous\_of\_second\_ptr == NULL)

{

head = first\_ptr;

}

else

{

previous\_of\_second\_ptr->next = first\_ptr;

}

node\* temp\_3;

temp\_3 = second\_ptr->next;

second\_ptr->next = first\_ptr->next;

first\_ptr->next = temp\_3;

}

};

int main()

{

LinkedList a;

bool check = true;

int choice, value\_1, value\_2;

while (check)

{

cout << "1) Insert value " << endl;

cout << "2) Enter node value you want to swap " << endl;

cout << "3) Display the values " << endl;

cout << "0) Exit" << endl;

cout << "Enter choice: ";

cin >> choice;

cout << endl;

switch (choice)

{

case 1:

cout << "Enter a vlaue: ";

cin >> value\_1;

a.insertion(value\_1);

cout << endl;

break;

case 2:

value\_1 = 0;

cout << "\nEnter node value which you want to swap : ";

cin >> value\_1;

cout << "\nEnter another node value which you want to swap : ";

cin >> value\_2;

a.swap(value\_1, value\_2);

cout << "List has been swapped!" << endl;

break;

case 3:

cout << "Link List is: ";

a.display();

cout << endl;

break;

case 0:

exit(0);

break;

default:

cout << "Invalid Choice " << endl;

break;

break;

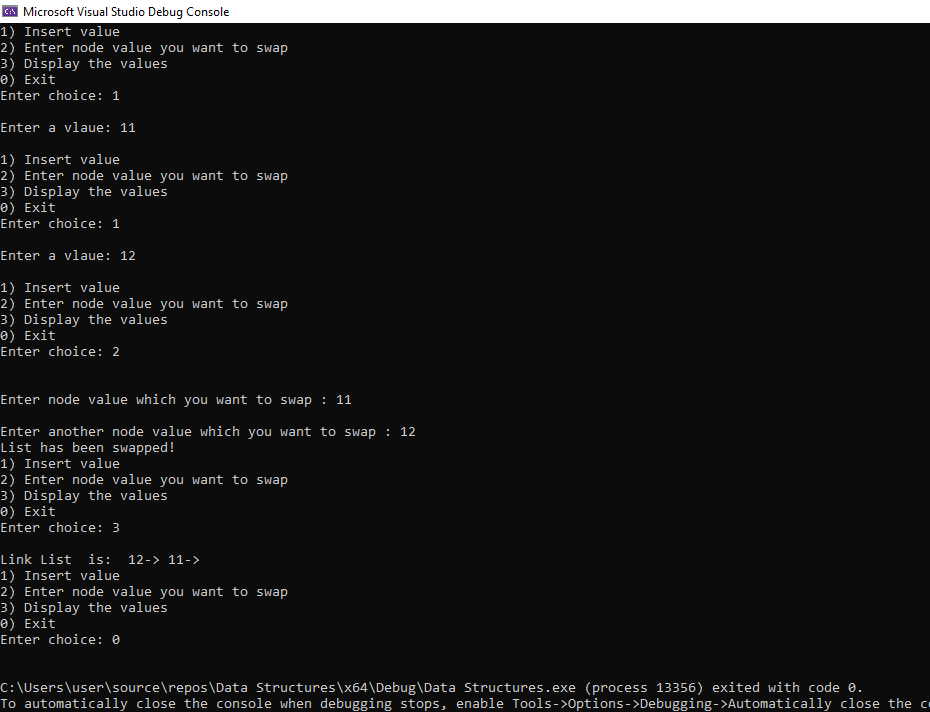
}

}

system("pause");

return 0;

}



**Task # 03**

#include<iostream>

using namespace std;

struct Node

{

int data;

Node\* next;

};

class LinkedList

{

public:

Node\* head;

Node\* tail;

LinkedList()

{

head = tail = NULL;

}

void insertion(int data)

{

Node\* new\_node = new Node;

new\_node->data = data;

new\_node->next = NULL;

if (head == NULL)

{

head = tail = new\_node;

}

else

{

tail->next = new\_node;

tail = tail->next;

}

}

void display()

{

Node\* temp\_1 = NULL;

temp\_1 = head;

while (temp\_1 != NULL)

{

cout << " " << temp\_1->data << "->";

temp\_1 = temp\_1->next;

}

cout << endl;

}

void ReverseLinkList()

{

Node\* current\_ptr, \* next\_ptr, \* previous\_ptr = 0;

current\_ptr = head;

next\_ptr = head;

while (next\_ptr != NULL)

{

next\_ptr = next\_ptr->next;

current\_ptr->next = previous\_ptr;

previous\_ptr = current\_ptr;

current\_ptr = next\_ptr;

}

head = previous\_ptr;

}

};

int main()

{

LinkedList a;

a.insertion(0);

a.insertion(99);

a.insertion(33);

a.insertion(66);

cout << "LL before: ";

a.display();

cout << endl;

cout << "LL after: ";

a.ReverseLinkList();

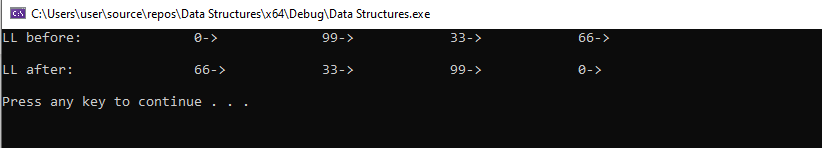
a.display();

cout << endl;

system("pause");

return 0;

}



**Task # 04**

#include<iostream>

#include<iomanip>

using namespace std;

struct node

{

int data;

node\* previous\_node;

node\* next\_node;

};

class doubly\_link\_list {

public:

node\* head;

node\* tail;

doubly\_link\_list()

{

head = tail = NULL;

}

void insertion(int value)

{

node\* new\_node = new node;

new\_node->data = value;

new\_node->next\_node = NULL;

new\_node->previous\_node = NULL;

if (head == NULL)

{

head = new\_node;

tail = new\_node;

}

else

{

tail->next\_node = new\_node;

new\_node->previous\_node = tail;

tail = tail->next\_node;

}

}

void display()

{

node\* temp\_1 = head;

while (temp\_1 != NULL)

{

cout << " " << temp\_1->data;

temp\_1 = temp\_1->next\_node;

}

}

void sum(int value)

{

bool found = false;

node\* temp\_1 = head;

while (temp\_1 != NULL)

{

node\* temp\_2 = temp\_1->next\_node;

while (temp\_2 != NULL)

{

if (temp\_1->data + temp\_2->data == value)

{

found = true;

cout << "The Sum is: ";

cout << "" << temp\_1->data << "," << temp\_2->data << " ";

cout << endl;

}

temp\_2 = temp\_2->next\_node;

}

temp\_1 = temp\_1->next\_node;

}

if (found == false)

{

cout << endl << "No Sum Found!" << endl;

}

}

};

int main()

{

doubly\_link\_list a;

bool check = true;

int choice, value\_1;

while (check)

{

cout << "1) Insert values in link list " << endl;

cout << "2) Which number you want to find sum of" << endl;

cout << "3) Display the values " << endl;

cout << "0) Exit" << endl;

cout << "Enter choice: ";

cin >> choice;

cout << endl;

switch (choice)

{

case 1:

cout << "Enter a vlaue: ";

cin >> value\_1;

a.insertion(value\_1);

cout << endl;

break;

case 2:

value\_1 = 0;

cout << "Enter node number which you want to find sum from link list: ";

cin >> value\_1;

a.sum(value\_1);

cout << endl;

break;

case 3:

cout << "Link List is: ";

a.display();

cout << endl;

break;

case 0:

exit(0);

break;

default:

cout << "Invalid Choice " << endl;

break;

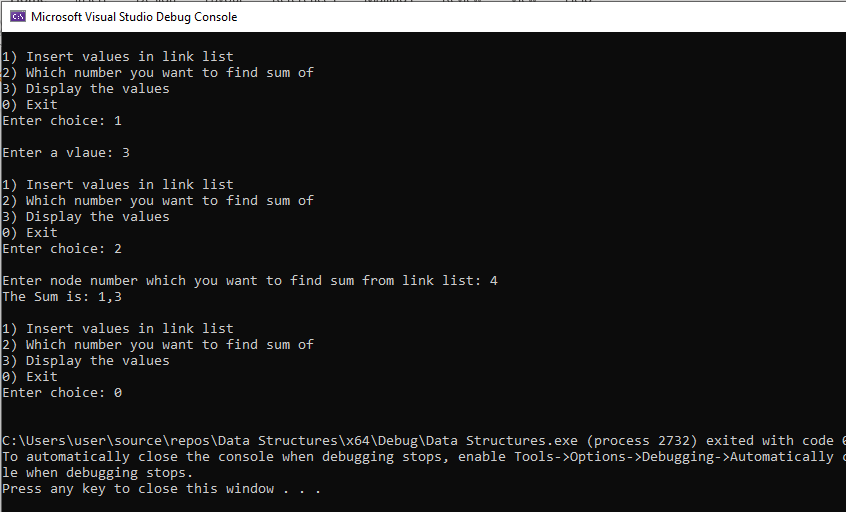
}

}

system("pause");

return 0;

}



**Stack**

**Task # 01**

#include<iostream>

#include<stack>

#include<map>

using namespace std;

map<int, int> frequencyMapper;

map<int, stack<int> > setMap;

int maxfreq = 0;

void push(int x)

{

int freqeuncy = frequencyMapper[x] + 1;

frequencyMapper[x] = freqeuncy;

if (freqeuncy > maxfreq)

maxfreq = freqeuncy;

setMap[freqeuncy].push(x);

}

int pop()

{

int top = setMap[maxfreq].top();

setMap[maxfreq].pop();

frequencyMapper[top]--;

if (setMap[maxfreq].size() == 0)

maxfreq--;

return top;

}

int main()

{

push(2);

push(3);

push(4);

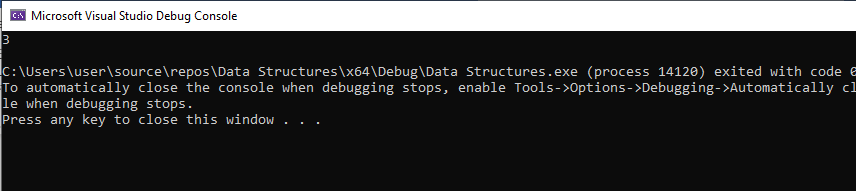
push(3);

push(5);

cout << pop() << "\n";

return 0;

}



**Task # 02**

**Task # 03**

**Queue**

**Task # 01**

#include<iostream>

#include<stack>

#include<queue>

using namespace std;

void reverseQueue(int k, queue<int>& Queue)

{

if (Queue.empty() == true || k > Queue.size())

return;

if (k <= 0)

return;

stack<int> St;

for (int i = 0; i < k; i++)

{

St.push(Queue.front());

Queue.pop();

}

while (!St.empty())

{

Queue.push(St.top());

St.pop();

}

for (int i = 0; i < Queue.size() - k; i++)

{

Queue.push(Queue.front());

Queue.pop();

}

}

void Print(queue<int>& Queue)

{

while (!Queue.empty())

{

cout << Queue.front() << " ";

Queue.pop();

}

}

int main()

{

queue<int> Queue;

Queue.push(1);

Queue.push(2);

Queue.push(3);

Queue.push(4);

Queue.push(5);

Queue.push(6);

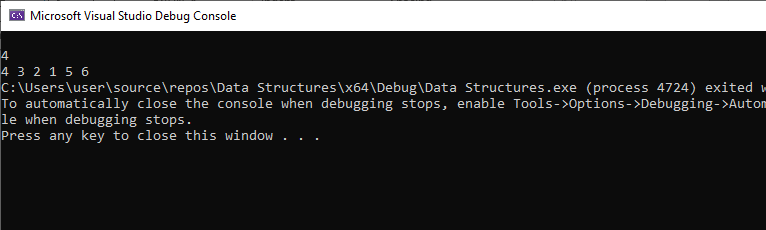
int k;

cin >> k;

reverseQueue(k, Queue);

Print(Queue);

}



**Task # 02**

#include<iostream>

using namespace std;

struct node

{

int data;

node\* next;

};

class Stack

{

public:

node\* top;

Stack()

{

top = NULL;

}

void push(int value)

{

node\* new\_node = new node;

new\_node->data = value;

new\_node->next = top;

top = new\_node;

}

void pop()

{

if (top == NULL)

{

cout << "Stack is empty" << endl;

}

else

{

node\* temp = top;

top = top->next;

temp->next = NULL;

delete temp;

}

}

void display()

{

if (top != NULL)

{

node\* temp\_2 = NULL;

temp\_2 = top;

while (temp\_2 != NULL)

{

cout << " -> " << temp\_2->data;

temp\_2 = temp\_2->next;

}

}

else if (top == NULL)

{

cout << "Stack is Empty!" << endl;

}

}

bool isEmpty()

{

if (top == NULL)

{

return true;

}

else

{

return false;

}

}

int Peek()

{

if (top == NULL)

{

return false;

}

else if (top != NULL)

{

int peek\_value = 0;

peek\_value = top->data;

return peek\_value;

}

}

};

class queue :public Stack

{

public:

Stack s1, s2;

void enqueue(int value)

{

while (!s1.isEmpty())

{

s2.push(s1.Peek());

s1.pop();

}

s1.push(value);

while (!s2.isEmpty())

{

s1.push(s2.Peek());

s2.pop();

}

}

int dequeue()

{

if (s1.isEmpty())

{

cout << "Queue is Empty!" << endl;

}

int val = s1.Peek();

s1.pop();

return val;

}

void Display()

{

if (s1.top != NULL)

{

node\* temp\_2 = NULL;

temp\_2 = s1.top;

while (temp\_2 != NULL)

{

cout << " -> " << temp\_2->data;

temp\_2 = temp\_2->next;

}

}

else if (s1.top == NULL)

{

cout << "Queue is Empty!" << endl;

}

}

};

int main()

{

queue a;

a.enqueue(3);

a.enqueue(6);

a.enqueue(9);

a.enqueue(12);

cout << endl << "Queue values using Stack: " << endl;

a.Display();

cout << endl;

cout << "The value Dequeue from the queue is: ";

cout << a.dequeue();

cout << endl;

cout << "Queue is: ";

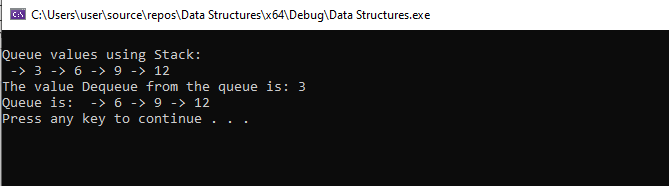
a.Display();

cout << endl;

system("pause");

return 0;

}



**Task # 03**

#include<iostream>

#include<map>

#include<queue>

using namespace std;

map<int, pair<string, string>>mp;

priority\_queue<pair<int, int> > pq, cpy;

int amt = 100000;

void login()

{

amt -= 400;

if (amt <= 0)

amt = 100000;

pair<int, int> check = pq.top();

cout << mp[check.second].first << ' ' << mp[check.second].second << ' ' << check.first << ' ' << amt << "\n";

pq.pop();

}

void Register()

{

string name, cnic;

int el;

cout << " Enter Name: ";

cin >> name;

cout << " Enter CNIC: ";

cin >> cnic;

cout << " Enter Emergency Level: ";

cin >> el;

mp[pq.size()] = { name,cnic };

pq.push({ el,pq.size() });

return;

}

void display()

{

cpy = pq;

while (!cpy.empty())

{

pair<int, int> check = cpy.top();

cout << mp[check.second].first << ' ' << mp[check.second].second << ' ' << check.first << "\n";

cpy.pop();

}

}

void menu()

{

int c = 0;

cout << "l. Login as Doctor" << endl;

cout << "2. Register for Checkup" << endl;

cout << "3. Display patient" << endl;

cout << "0. Exit" << endl;

cout << "Enter choice: ";

cin >> c;

switch (c)

{

case 1:

login();

menu();

break;

case 2:

Register();

menu();

break;

case 3:

display();

menu();

break;

case 0:

exit(0);

break;

}

}

int main()

{

menu();

}

